Part I

TO-DOs

a) The mean is a Dx1 vector calculated individually for each dimension of data. The cov matrix (DxD) is calculated from combinations of covariances between each 1-D vector.

b) Gaussian likelihood is calculated using the expression for multivariate normal probability density function. It takes multidimensional parameters mean and covariance, and one data example.

c) Using Bayes rule, we can obtain the posterior for Pr(h=1|x):
likeSkin * priorSkin / (likeSkin * priorSkin + likeNonSkin * priorNonSkin) =
= Pr(x|h=1) * Pr(h=1) / ∑{k} [Pr(x|h=k) * Pr(h=k)] =
= Pr(h=1|x)

d) Matlab's *randn* function generates a normally distributed random value, around the mean 0, and with variance 1. By multiplying the obtained value by our variance, and then adding up the mean, we get a a random sample, which is normally distributed around our parameters.

e) Calculating likelihood, as in b), but in a mixture of gaussians. Each gaussian is associated with a weight, and therefore the likelihood for each gaussian is multiplied by that weight, and then is added to the likelihoods for the same data item.

f) Drawing vectors from a normal distribution, as in d), but now with 3-D vectors. Since the covariance is multidimensional, we have to apply a Cholesky factorization to it first, so that we can use this value to manipulate *randn* as we wish. As before, add up the mean vector in the end.

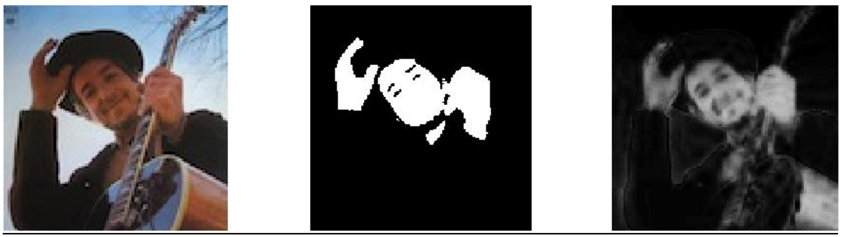
g) The probability Pr(h_i=k|x_i,theta) that the kth gaussian was responsible for the ith data point.

h) Updating weights of each gaussian based on the relative responsibility of each component for the data points.

i) Updating the mean associated with the kth gaussian based on the vectors that the kth component "explains" the best (the ones it is the most responsible for).

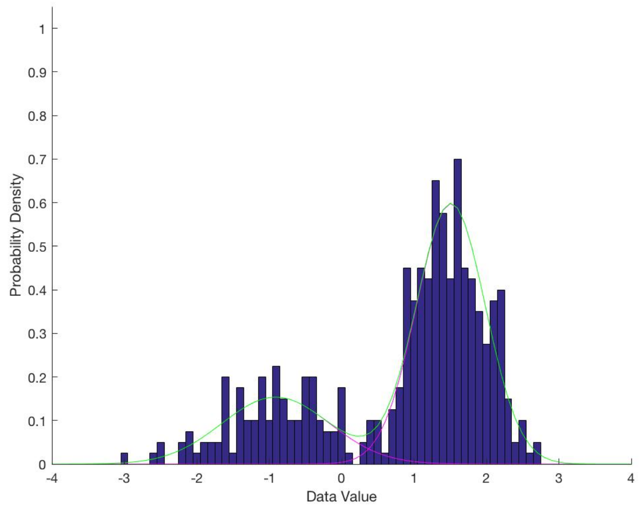
j) Similarly to (h) and (i), we're updating the kth covariance to better explain the data points more associated with the kth gaussian.

Part A



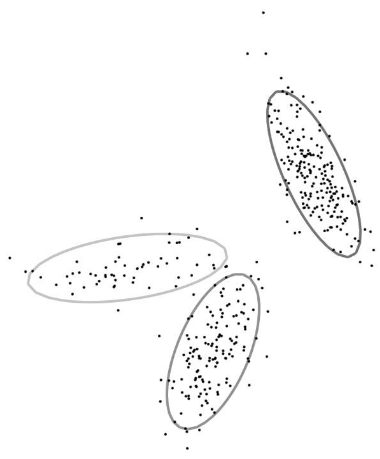
As the skin as a relatively uniform tone, most of the skin was correctly identified as being skin. Because the algorithm has only one mean, darker areas like facial hair or shadows cannot be overgeneralized to be seen as skin. As the gaussian also only has information about the color, similar materials like wood will be misclassified as skin.

Part B



The algorithm did a great job retrieving the original parameters. The slight differences are due to the randomness in the small amount of data.

Part C



The algorithm also did a great job retrieving the original parameters. In this case it's easier that the random initialization leads to a local minima.

Part II

I trained two mixtures of Gaussians, one for detecting apple pixels, and another for detecting non-apple pixels. The data was three-dimensional, and was derived from the pictures in the folder apples.zip. I used two of the images as the training set, and the last as validation set.
To model both mixtures of Gaussians, I divided the labeled apple pixels from the non-apple pixels in the training set. This gave me several 3*I matrices, some for apple=true, and others for apple=false. I then concatenated all pixels into RGBApple or RGBNonApple, but left one image of the training set out of this process. I used this image to create the validation set.
I used a prior of 0.3 for Apple, and 0.7 for NonApple. I decided to dedicate 2 gaussians to model the Apple pixels, and 4 gaussians to model Non-Apple pixels. To get to these values, I recorded the F-1 score from True Positives, False Positives, True Negatives and False Negatives, when evaluating the validation set (*function posterior*) with the parameters obtained from the training set (*function fitMixGauss*). The following table has my results:

F-1 scores		**K for NonApple MoG**				
		1	2	3	4	5
K for Apple MoG	1	NaN	0.4	0.7	0.8	0.9
	2	NaN	NaN	0.7	0.9	0.8
	3	NaN	0.1	0.7	0.9	0.9
	4	0.2	0.3	0.4	0.8	0.9

Since computing 5 gaussians is more computationally expensive, I chose the options 2 and 4, for apple and non-apple pixels, respectively. Having more than one gaussian allowed me to get a flexible sets of means, in order to encompass the different types of apples, and especially well the noisy backgrounds. Things like pears and tangerines will not be correctly classified, because of the similarities in color and brightness with apples. I set a minimum of 0.02 for the weight of a given gaussian so that the algorithm didn't collapsed to a mixture of fewer gaussians, and also to prevent errors in Matlab when dealing with infinitesimally small values.

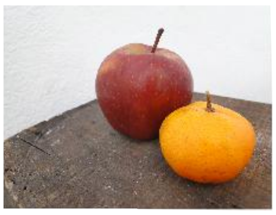
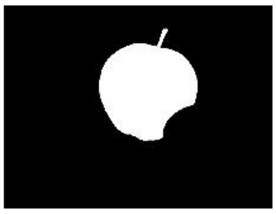
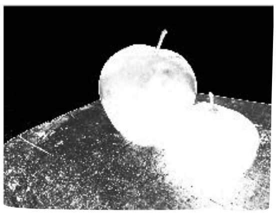
I resized all training pictures by a factor of 0.2, and the test pictures by 0.5, to enhance the speed of processing. Virtually all the information about contours and the identity of objects is not lost in this process. I also tried to parallelize the code, but I think the functions I used to convert and act on cells may not be taking advantage of parallelism. The code doesn't seem faster.

Finally, I trained the gaussians with the E-M algorithm. With the two newly generated mixGaussEst, I calculated the posterior probability of each pixel in the test images (testApple.zip) being part of an apple, and then plotted these graphs.

These are the results for the test examples:

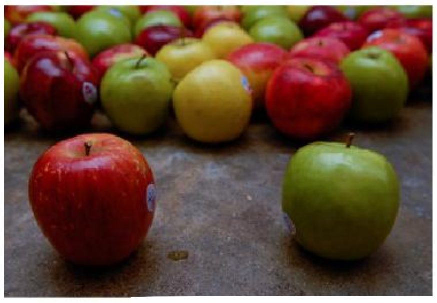
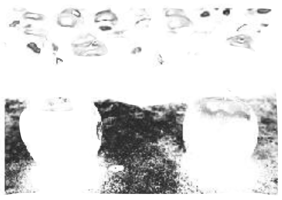
testApples.zip:

- Bbr98ad4z0A-ctgXo3gdwu8-original

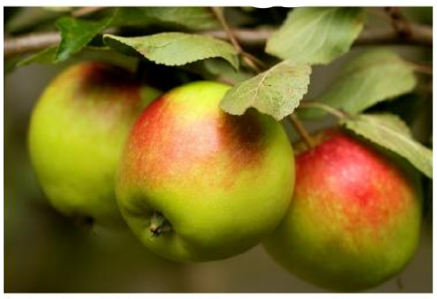
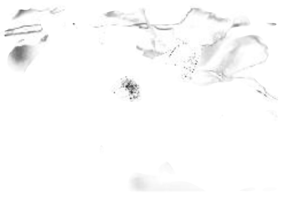




F-1 score: 0.511 (TP: 95503, TN: 471439, FP: 182496, FN: 562, precision: 0.344, recall: 0.994)

- Apples_by_MSR_MikeRyan_flickr

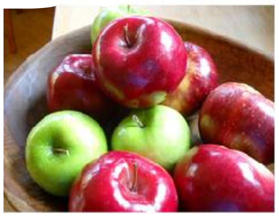
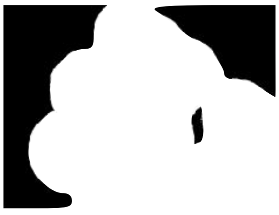
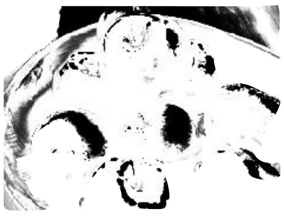



- audioworm-QKUJj2wmxuI-original

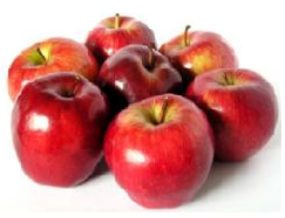
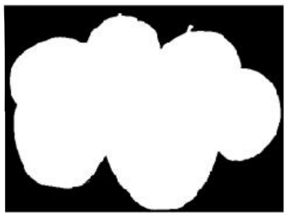
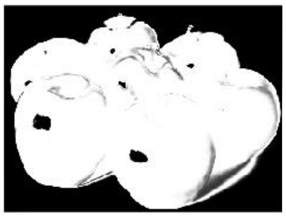
My images:

- more-apples

F-1 score: 0.771 (TP: 26853, TN: 4217, FP: 11520, FN: 4410, precision: 0.700, recall: 0.859)

- red-apples

F-1 score: 0.969 (TP: 30899, TN: 17033, FP: 1051, FN: 937, precision: 0.967, recall: 0.971)